[MS-DRM]:
Digital Rights Management License Acquisition Data Structure

Revision Summary

Date	Revision History	Revision Class	Comments
05/11/2007	0.1		MCPP Milestone 4 Initial Availability
08/10/2007	1.0	Major	Updated and revised the technical content.
09/28/2007	1.0.1	Editorial	Revised and edited the technical content.
10/23/2007	1.0.2	Editorial	Revised and edited the technical content.

Date	Revision History	Revision Class	Comments
11/30/2007	1.0.3	Editorial	Revised and edited the technical content.
01/25/2008	1.0.4	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction

The Digital Rights Management License Acquisition Protocol provides secure distribution, promotion, and sale of digital media content. The protocol is used to acquire licenses for Windows Media content by using **Digital Rights Management** version 1, Digital Rights Management version 7, or Digital Rights Management version 11 technologies.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

Big-Endian
Globally Unique Identifier (GUID)
Little-Endian

The following terms are specific to this document:

Cryptographic operator "K{text}": Text encrypted with symmetric key K. For more information, see [NSPCPW].

Cryptographic operator "[text]$_K$": Text signed with private portion of asymmetric key K, K_{priv}. For more information, see [NSPCPW].

Cryptographic operator "{text}$_K$": Text encrypted with public portion of asymmetric key K, K_{pub}. For more information, see [NSPCPW].

Digital Rights Management (DRM): A set of technologies that provides control over how a given piece of protected content may be used.

Elliptic curve cryptography (ECC): A public-key crypto system that is based on high-order elliptic curves over finite fields.

Mathematical operator "⊕": A bitwise exclusive OR. For more information, see [NSPCPW].

Mathematical operator "~": A bitwise negation.

Mathematical operator "|": A concatenation. For more information, see [NSPCPW].

Rivest Cipher 4 (RC4): RSA symmetric key encryption algorithm. **RC4** is a proprietary encryption algorithm available under license from RSA Security, as specified in [RC4-ENCRYPT].

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

[RC4-ENCRYPT] RSA Security, "RC4 Encryption Operation", https://www.rsa.com/products/bsafe/documentation/cryptoc62html/group__ALG__RC4__EXAMPLE.html

[RFC2045] Freed, N., et al., "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996, http://ietf.org/rfc/rfc2045.txt

[RFC2109] Kristol, D., and Montulli, L., "HTTP State Management Mechanism", RFC 2109, February 1997, http://www.ietf.org/rfc/rfc2109.txt

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

[RFC2616] Fielding, R., et al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999, http://www.ietf.org/rfc/rfc2616.txt

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000, http://www.ietf.org/rfc/rfc2818.txt

[RFC2821] Klensin, J., "Simple Mail Transfer Protocol", RFC 2821, April 2001, http://www.ietf.org/rfc/rfc2821.txt

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006, http://www.ietf.org/rfc/rfc4648.txt

[RSAFAQ] RSA Laboratories, "Frequently Asked Questions About Today's Cryptography, Version 4.1", May 2000, http://www.rsa.com/rsalabs/faq/files/rsalabs_faq41.pdf

[XML] World Wide Web Consortium, "Extensible Markup Language (XML) 1.0 (Fourth Edition)", W3C Recommendation, September 2006, http://www.w3.org/TR/REC-xml

[XMLSCHEMA1/2] Thompson, H.S., Ed., Beech, D., Ed., Maloney, M., Ed., and Mendelsohn, N., Ed., "XML Schema Part 1: Structures Second Edition", W3C Recommendation, October 2004, http://www.w3.org/TR/xmlschema-1/

[XMLSCHEMA2/2] Biron, P.V., Ed. and Malhotra, A., Ed., "XML Schema Part 2: Datatypes Second Edition", W3C Recommendation, October 2004, http://www.w3.org/TR/xmlschema-2

1.2.2 Informative References

[CAECCRYPT] Barbosa, M., Moss, A., and Page, D., "Compiler Assisted Elliptic Curve Cryptography", http://eprint.iacr.org/2007/053.pdf

[ELLIPTICCURVE] RSA Laboratories, "Overview of Elliptic Curve Cryptosystems", June 1997, http://www.rsa.com/rsalabs/node.asp?id=2013

[ELLIPTICCURVE-DSA] Farkas, S., "Elliptic Curve DSA", http://blogs.msdn.com/shawnfa/archive/2007/01/18/elliptic-curve-dsa.aspx

[MSDN-WMF11SDK] Microsoft Corporation, "Windows Media Format 11 SDK", http://msdn2.microsoft.com/en-us/library/aa387410.aspx

[MSDN-WMRMHEADOBJ] Microsoft Corporation, "WMRMHeader Object", http://msdn2.microsoft.com/en-us/library/ms984909.aspx

[NSPCPW] Perlman, R., Speciner, M., and Kaufman, C., "Network Security: Private Communication in a Public World", New York, 1980, ASIN: B000N7EJQQ.

[SCHNEIER] Schneier, B., "Applied Cryptography, Second Edition", John Wiley and Sons, 1996, ISBN: 0471117099.

If you have any trouble finding [SCHNEIER], please check here.

[SDMI] SDMI, (SECURE DIGITAL MUSIC INITIATIVE) "SDMI Portable Device Specification Part 1 Version 1.0", July 8th 1999, http://ntrg.cs.tcd.ie/undergrad/4ba2.01/group10/port_device_spec_part1.pdf

[X9.62] American National Standards Institute, "Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)", ANSI X9.62:2005, 2005, http://webstore.ansi.org/ansidocstore/product.asp?sku=ANSI+X9%2E62%3A2005

Note There is a charge to download the specification.

1.3 Protocol Overview (Synopsis)

The Digital Rights Management License Acquisition Protocol version 1, version 7, and version 11 provide a means of acquiring a license for Windows Media content.

When using Digital Rights Management License Acquisition Protocol version 1, the client generates a license request and sends it to a license server as an HTTP GET request. The server receives the GET request and returns the license to the client as part of an HTML page containing an Internet Explorer ActiveX control or a Netscape plug-in responsible for storing the license in the client.

The Digital Rights Management License Acquisition Protocol version 7 uses a packet containing a license request in extensible markup language (XML) format and is sent using an HTTP POST request. The server responds with an XML packet containing any number and combination of version 1 and version 7 licenses.

The Digital Rights Management License Acquisition Protocol version 11 is functionally equivalent to the version 7 protocol, with the addition of a few XML fields in the license request challenge body.

1.3.1 Digital Rights Management Version 1

The Digital Rights Management License Acquisition Protocol version 1 provides the means of acquiring a license for Windows Media content. Its packets include a client request for a license and a server response that contains the license.

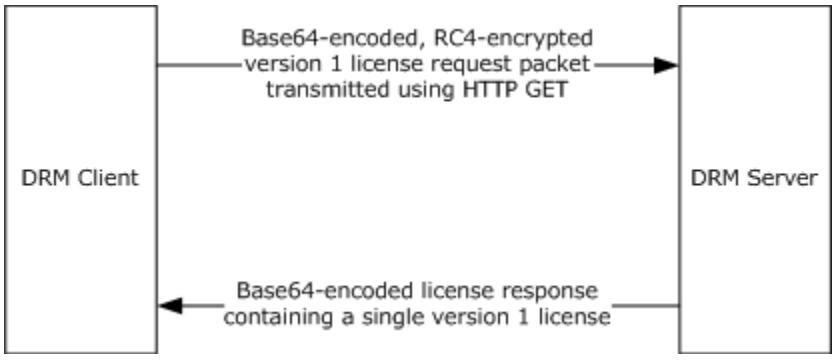

Figure 1: DRM version 1 license request and response

The Digital Rights Management client application generates a license request and sends it to a license server. The request is a binary string that is partially encrypted using the **Rivest Cipher 4**

(RC4) (as specified in [RC4-ENCRYPT]) and then encoded using the Base64 Encoding algorithm, as specified in section 2.2.1.1.

The response is a single version 1 license, formatted as a binary string, and encoded with the base64 encoding algorithm, as specified in section 2.2.1.1. It is returned to the client as part of an HTML page that contains an Internet Explorer ActiveX control or a Netscape plug-in.

A Digital Rights Management License Acquisition Protocol version 1 license is represented as specified in section 2.3.3.

The structures that are used by version 1, version 7 and version 11 of Digital Rights Management License Acquisition Protocol are specified in section 2.2.1.

This protocol uses the following packets.

Packet	Description
DRM_Version 1 License Request	Contains the client's request for a license.
DRM_Version 1 License Response	Contains the server's response to the client's request for a license.

RC4 is a proprietary encryption algorithm available under license from RSA Security, as specified in [RSAFAQ].

1.3.2 Digital Rights Management Version 7

The Digital Rights Management License Acquisition Protocol version 7 provides the means of acquiring a license for Windows Media content. Its packets include a client request for a license and a server response that contains the license.

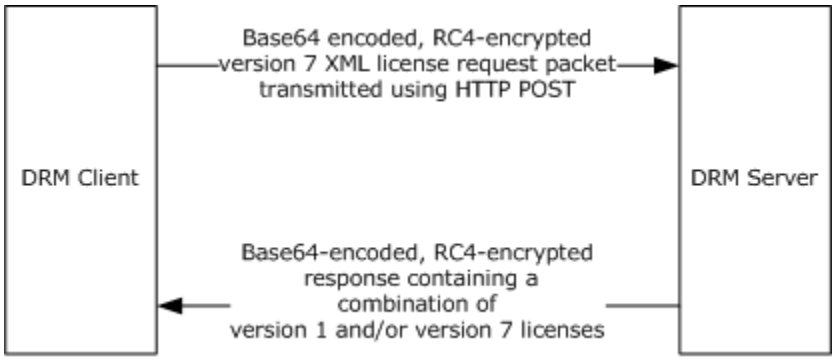

Figure 2: DRM version 7 license request and response

The Digital Rights Management client generates a license request and sends it to a license server. The request is in extensible markup language (XML) format, partially RC4-encrypted, and then encoded using the base64 algorithm, as specified in section 2.2.1.1. It is sent to the server by means of an HTTP POST request. For more information about RC4, see Remarks at the end of this topic.

The response is an RC4-encrypted XML packet. The first 40 bytes of the response packet are represented by an **ECC**-encrypted RC4 key. The remainder of the packet (the license data itself) is encrypted with the provided RC4 key. The packet is then encoded with the base64 algorithm, as specified in section 2.2.1.1. It can contain any number and combination of version 1 and version 7

licenses. Each version 7 license is itself RC4-encrypted using the mechanism described in section 1.3.2.

A Digital Rights Management License Acquisition Protocol version 7 license is represented, as specified in section 2.4.4.

The structures that are used by both version 1 and version 7 of the Digital Rights Management License Acquisition Protocol are specified in section 2.2.1.

This protocol uses the following packets.

Packet	Description
DRM Version 7 License Request	Contains the client's request for a license.
DRM Version 7 License Response	Contains the server's response to the client's request for a license.
DRM Version 7 License Format	Contains an XML-formatted license.

RC4 is a proprietary encryption algorithm available under license from RSA Security, as specified in [RSAFAQ].

1.3.3 Digital Rights Management Version 11

The Digital Rights Management License Acquisition Protocol version 11 is almost identical to the version 7 protocol, with the addition of a few fields in the license request packet.

This protocol uses the following packets.

Packet	Description
DRM Version 11 License Request	Contains the client's request for a license.
DRM Version 11 License Response	Contains the server's response to the client's request for a license.
DRM Version 11 License Format	Contains an XML-formatted license.

1.4 Relationship to Other Protocols

The following are technologies that users should be very familiar with before attempting Digital Rights Management license acquisition:

- HTTP
- HTTPS (TLS or SSL)
- XML
- XSD

1.5 Prerequisites/Preconditions

The following data must be licensed from Microsoft for the license acquisition server prior to implementing any of these protocols:

- Private server cryptographic key (KS_{priv}).

- Server certificate chain (CS), including a certificate representing the root issuer (CR).
- Client application versions and associated PUBKEYs for verification of the client certificate (CA).

The following keys and certificates are used by the client application and referenced in this document:

- Private client cryptographic key (KC_{priv}).
- Public server cryptographic key (KS_{pub}).
- Client application certificate (CA) (leaf certificate only).

1.6 Applicability Statement

None.

1.7 Versioning and Capability Negotiation

In the Digital Rights Management License Acquisition Protocol, there is no facility for version or capability negotiation. The client must submit requests to a server that understands the maximum protocol version used by the client. In practice, content providers embed license acquisition specifics within the content file headers. This information indicates to the client which license version and license acquisition protocol will be used.

This protocol can be implemented on top of the following:

- TCP
- HTTP
- HTTPS

1.8 Vendor-Extensible Fields

Within the version 7 and version 11 license response packet, vendors are free to add any well-formed XML data to the <META> element. The contents of this element are not used by the Digital Rights Management client application.

1.9 Standards Assignments

None.

2 Messages

The following sections specify how Digital Rights Management License Acquisition Protocol messages are transported and Digital Rights Management License Acquisition Protocol message syntax.

2.1 Transport

The Digital Rights Management License Acquisition Protocol MAY use HTTP (as specified in [RFC2616]) or HTTP over TLS (as specified in [RFC2818]) as the transport layer.<1> The use of HTTP over TLS is triggered by the specification of an "https" URL, rather than an "http" URL. Messages and data are sent via URI query strings, HTTP POST headers, and HTTP responses.

Some client applications may also use the HTTP cookie mechanism (as specified in [RFC2109]) as a transport and state management mechanism outside the purview of license acquisition. The HTTP cookie mechanism allows named data items to be sent from one party to another as part of an HTTP message, stored by the receiving party, and returned automatically to the original party as part of all subsequent HTTP messages to that party.

2.2 Message Syntax

2.2.1 Common Data Types and Algorithms

The following structures and algorithms are common to version 1, version 7 and version 11 of the Digital Rights Management License Acquisition Protocol.

Unless otherwise noted, all multi-octet integral values are stored in **little-endian** format.

Unless otherwise noted, all data structures are packed to 4-octet alignment.

For more information about encryption algorithms within this document, see [CAECCRYPT], [ELLIPTICCURVE], [ELLIPTICCURVE-DSA], [SCHNEIER] section 19.6, and [X9.62].

2.2.1.1 Base64 Encoding

The standard base64 encoding algorithm (as specified in [RFC4648]) is used to transmit binary data. Base64 processes data as 24-bit groups, mapping it to four encoded characters of 6 bits each. It is sometimes referred to as 3-to-4 encoding. Each 6-bit group in the 24-bit group is used as an index into a mapping table (see the following table) to obtain a character for the encoded data. By convention, line lengths in the encoded data are limited to 76 characters, but this is not strictly enforced in this protocol.

Note The characters used in base64 encoding do not include any of the special characters of the Simple Mail Transfer Protocol (SMTP) (as specified in [RFC2821]), or the hyphen used with Multipurpose Internet Mail Exchange (MIME) boundary strings, as specified in [RFC2045].

2.2.1.1.1 Base64 Mapping Table

0 A	17 R	34 i	51 z
1 B	18 S	35 j	52 0
2 C	19 T	36 k	53 1
3 D	20 U	37 l	54 2
4 E	21 V	38 m	55 3
5 F	22 W	39 n	56 4
6 G	23 X	40 o	57 5
7 H	24 Y	41 p	58 6

```
8 I     25 Z     42 q     59 7
9 J     26 a     43 r     60 8
10 K    27 b     44 s     61 9
11 L    28 c     45 t     62 +
12 M    29 d     46 u     63 /
13 N    30 e     47 v
14 O    31 f     48 w
15 P    32 g     49 x
16 Q    33 h     50 y
```

2.2.1.1.2 Example: base64 Encoding of 3 Bytes: "XYZ"

```
Input data          X        Y        Z
Input bits   01011000-01011001-01011010
Bit groups   010110-000101-100101-011010
Mapping           W      F      l      a
```

2.2.1.1.3 Base64 and DRM

In the Digital Rights Management License Acquisition Protocol, base64 encoding refers to a slightly modified version of the standard base64 algorithm. Digital Rights Management base64 encoding is identical to standard base64 encoding, with the exception of the last two characters in the following mapping table.

```
0 A     17 R     34 i     51 z
1 B     18 S     35 j     52 0
2 C     19 T     36 k     53 1
3 D     20 U     37 l     54 2
4 E     21 V     38 m     55 3
5 F     22 W     39 n     56 4
6 G     23 X     40 o     57 5
7 H     24 Y     41 p     58 6
8 I     25 Z     42 q     59 7
9 J     26 a     43 r     60 8
10 K    27 b     44 s     61 9
11 L    28 c     45 t     62 !
12 M    29 d     46 u     63 *
13 N    30 e     47 v
14 O    31 f     48 w
15 P    32 g     49 x
16 Q    33 h     50 y
```

2.2.1.2 Cryptographic Parameters

The following 160-bit elliptic curve cryptography (ECC) curves are used in this document:

ECC_1

Parameter	Value
p(modulus)	0x89abcdef012345672718281831415926141424f7
a	0x37a5abccd277bce87632ff3d4780c009ebe41497
b	0x0dd8dabf725e2f3228e85f1ad78fdedf9328239e
generator x	0x8723947fd6a3a1e53510c07dba38daf0109fa120
generator y	0x445744911075522d8c3c5856d4ed7acda379936f
curve order	0x89abcdef012345672716b26eec14904428c2a675

Prior to encryption, the plaintext (length 1 – 16 bytes) is prepared with the following sequence of operations.

1. Copy the plaintext into a buffer, "x", comprised of five DWORDs.
2. The fifth DWORD of x is set to zero.
3. If there is a solution for y in the following equation, then x||y is now ready for encryption.

```
(y^2) mod p = (x^3 + ax + b) mod p
```

4. If there is no solution to this equation, increment the fifth DWORD of x and repeat the previous step.

2.2.1.3 Cryptographic Keys

The client and server use a set of cryptographic keys as follows:

KC: An ECC_1 key that represents the client application. The client knows KC_{priv} and the server knows KC_{pub}.

KS: An ECC_1 key that represents the server. The client knows KS_{pub} and the server knows KS_{priv}.

KM: An ECC_1 key that represents a specific instance of a machine running the client application. KM_{pub} is transmitted from the client to server during a license request.

2.2.1.4 PK

The **PK** structure contains the **PUBKEY** structure and its version information.

```
typedef struct {
  PUBKEY pubkey;
  BYTE version[4];
} PK;
```

pubkey: A **PUBKEY** structure that contains a public key.

version: A 4-byte buffer that contains version information for the public key. MUST be {0x00, 0x01, 0x00, 0x00}.

2.2.1.5 PKCERT

The **PKCERT** structure contains a signed certificate.

```
typedef struct {
  PK pk;
  BYTE sign[40];
} PKCERT;
```

pk: A **PK** structure that contains a public key and its version information.

sign: A 40-byte buffer that contains the signature of the pk member. This signature is created using ECDSA over curve ECC_1. For more information about ECDSA, see [ELLIPTICCURVE-DSA].

$[pk]_K$

where K is an ECC_1 key.

2.2.1.6 PUBKEY

The **PUBKEY** structure contains a public key.

```
typedef struct {
  BYTE y[40];
} PUBKEY;
```

y: A 40-byte buffer that contains a public key. This is the public portion of a public/private key pair in ECC_1. The x-coordinate is stored in bytes 0 - 19; the y-coordinate in bytes 20 - 39. The two coordinates are base 0x100000000 integers stored in little-endian order.

2.3 DRM Version 1 Data Types

The following structures and algorithm are specific to version 1 of the Digital Rights Management License Acquisition Protocol.

2.3.1 DRM Version 1 License Request

The DRM Version 1 License Request packet is used by the client to request a license for Windows Media content. This packet is transmitted to the server via URI parameter "challenge" as a Digital Rights Management (DRM) base64-encoded value. The URI parameter "DRMVer=1.4" is also sent to the server with this license request. Note that the value for "DRMVer" may vary based on the client application.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Version																															
EncRandNum																															
...																															
...																															
...																															
...																															
...																															
...																															
...																															
...																															
(EncRandNum cont'd for 12 rows)																															
pkcert																															
...																															
...																															
...																															
...																															
...																															
...																															
...																															
(pkcert cont'd for 13 rows)																															

KeyID
...
...
...
...
...
...
Rights
AppSec

Version (4 bytes): Request version. MUST be {0x00, 0x01, 0x00, 0x01}.

EncRandNum (80 bytes): One-time use random number encrypted using ECC_1 with the public server cryptographic key (KS). The first 7 bytes (unencrypted) of **EncRandNum** are used as the initialization vector (IV) to create an RC4 key (KR).

pkcert (84 bytes): RC4-encrypted PKCERT containing a signed copy of KM_{pub}.

KeyID (28 bytes): RC4-encrypted content key identifier. The content key ID is generated by the server using Windows Media Rights Manager (WMRM) and stored in the header of a protected content stream. Only the first 25 bytes of this field are used.

Rights (4 bytes): RC4-encrypted requested playback rights, which can be any combination of the following values:

Byte Array	Meaning
{0x01, 0x00, 0x00, 0x00} 0x01000000	Right to play back content. This is also known as RIGHT_PLAY_ON_PC.
{0x02, 0x00, 0x00, 0x00} 0x02000000	Right to copy licensed content to a device that is not compliant with the Secure Digital Music Initiative (SDMI). This is also known as RIGHT_COPY_TO_NONSDMI_DEVICE.
{0x08, 0x00, 0x00, 0x00} 0x08000000	Right to copy licensed content to a CD. This is also known as RIGHT_BURN_TO_CD.
{0x10, 0x00, 0x00, 0x00} 0x10000000	Right to copy licensed content to an SDMI device. This is also known as RIGHT_COPY_TO_SDMI_DEVICE.

AppSec (4 bytes): RC4-encrypted security level of the application making the request. The security level MUST be equal to the security level in the client application certificate (CA).

Cryptographic sequence:

1. pkcert.pk = KM_{pub}
2. pkcert.sign = $[pkcert.pk]_{KC}$
3. $\{EncRandNum\}_{KS}$
4. KR {pkcert}
5. KR {KeyID}
6. KR {Rights}
7. KR {AppSec}

2.3.2 DRM Version 1 License Response

The DRM Version 1 License Response takes the form of a Web page that uses a COM object to store a base64-encoded CERTIFIED_LICENSE structure in the client's local license store. The base64-encoded CERTIFIED_LICENSE is returned embedded in a script section of the Web page.

The normal sequence of operations taken by the script in the Web page to store the license locally is as follows:

1. Create an IWMDRMProvider instance via WMDRMCreateProvider or WMDRMCreateProtectedProvider.
2. Create an IWMDRMLicenseManagement instance by calling IWMDRMProvider::CreateInstance, passing in the IID of IWMDRMLicenseManagement.
3. Call IWMDRMLicenseManagement::StoreLicense to store the license locally.

For more information about these interfaces and functions, see [MSDN-WMF11SDK].

2.3.3 DRM Version 1 License Format

A Digital Rights Management (DRM) version 1 license is represented as a **CERTIFIED_LICENSE** structure, encoded with the base64 encoding algorithm and visible as a string. The structure consists of two certificates and a license. The first certificate represents the Microsoft signing certificate. The second certificate represents the signing certificate of the server issuing the content license.

2.3.3.1 Structures

The Digital Rights Management version 1 license format contains the following top-level structures.

Structure	Description
CERT	Defines the certificate component of a DRM version 1 certified license.
CERTDATA	Defines the data block of a certificate, including the public key, serial number, and certificate issuer.

Structure	Description
CERTIFIED_LICENSE	Defines a version 1 certified license before it is encoded with base64 encoding.
LICENSE	Defines the license portion of a version 1 certified license.
LICENSEDATA	Defines the data portion of a version 1 license, including the rights and security settings.

2.3.3.1 CERT

The **CERT** structure defines the certificate component of a Digital Rights Management (DRM) version 1 certified license.

```
typedef struct {
  BYTE certVersion[4];
  BYTE dataLen[4];
  BYTE sign[40];
  CERTDATA cd;
} CERT;
```

certVersion: A 4-byte buffer that contains the certificate version. Valid values for certificate version are {0x00, 0x01, 0x00, 0x00}.

dataLen: A 4-byte buffer that contains the size of the **cd** field in bytes, as a sequence of four hexadecimal values (this is a DWORD stored in **big-endian** order). For example, if cd is 300 bytes (0x12c bytes), this field contains {0x00, 0x00, 0x01, 0x2C}.

sign: A 40-byte buffer that contains the signature of the **cd** member. This signature is created using ECDSA over curve ECC_1. The key used to sign this data is the private key of the issuing authority.

$[cd]_K$

cd: A CERTDATA structure that contains the certificate data, including its public key, issuer and expiration date.

2.3.3.2 CERTDATA

The **CERTDATA** structure defines the data block of a certificate, including the public key, serial number, and certificate issuer.

```
typedef struct {
  BYTE pk[40];
  BYTE expiryDate[4];
  BYTE serialNumber[4];
  BYTE issuer[4];
  BYTE subject[4];
} CERTDATA;
```

pk: A 40-byte buffer that contains a public key. This is the public portion of a public/private key pair in ECC_1. The x-coordinate is stored in bytes 0 – 19; the y-coordinate in bytes 20 – 39.

expiryDate: A 4-byte buffer that contains the date on which the certificate expires. All values are encoded as hexadecimal. The first byte contains the value of the first two digits of the year, the second contains the value of the latter two digits of the year, the third contains the value of the month, and the fourth contains the value of the day. For example, the date 12/30/2002 is represented as {0x14, 0x02, 0x0C, 0x1E}.

serialNumber: A serial number that identifies the certificate.

issuer: A certificate server identifier.

subject: A number that identifies the subject of the certificate.

2.3.3.3 CERTIFIED_LICENSE

The **CERTIFIED_LICENSE** structure defines a version 1 certified license.

```
typedef struct {
  LICENSE license;
  CERT cert1;
  CERT cert2;
} CERTIFIED_LICENSE;
```

license: A **LICENSE** structure that contains the license component of a version 1 certified license.

cert1: A **CERT** structure that contains the Microsoft-signed root certificate representing the certificate server.

cert2: A **CERT** structure that contains the certificate server-signed certificate representing the license server.

2.3.3.4 LICENSE

The **LICENSE** structure defines the license portion of a version 1 certified license.

```
typedef struct {
  BYTE licVersion[4];
  BYTE dataLen[4];
  BYTE sign[40];
  LICENSEDATA ld;
} LICENSE;
```

licVersion: A 4-byte buffer that contains the license version. This value MUST contain {0x00, 0x01, 0x00, 0x00}.

dataLen: A 4-byte buffer that contains the size of the **ld** field in bytes, as a sequence of four hexadecimal values (this is a DWORD stored in big-endian order). For example, if **ld** is 300 bytes (0x12c bytes), this field contains {0x00, 0x00, 0x01, 0x2C}.

sign: A 40-byte buffer that contains the signature of the **ld** member. This signature is created using ECDSA over curve ECC_1. The key used to sign this data is the private key of the license server (KS).

ld: A **LICENSEDATA** structure that contains the license data, including the digital rights and security data.

Cryptographic Sequence:

sign = $[ld]_{KS}$

2.3.3.5 LICENSEDATA

The **LICENSEDATA** structure defines the data portion of a version 1 license, including the rights and security settings.

```
typedef struct {
  char KID[25];
  BYTE key[80];
  BYTE rights[4];
  BYTE appSec[4];
  BYTE expiryDate[4];
} LICENSEDATA;
```

KID: A 25-character array that contains the content key ID.

key: An 80-byte buffer that contains the encrypted RC4 content key ($K_{content}$) and a copy of its bitwise negation ($P_{content} = \sim K_{content}$). This field is encrypted using ECC_1 with KM. Prior to encryption and after decryption, bytes 0 – 6 of the plaintext represent $K_{content}$ and bytes 7 – 13 of the plaintext represent $P_{content}$. These values may be compared to ensure that they were stored and transmitted properly by calculating

$\sim(K_{content} \oplus P_{content})$

If this value is not 0, $K_{content}$ and/or $P_{content}$ are suspect and should not be used.

rights: A 4-byte buffer that contains the client rights for the licensed content. These values are logically combined in byte order.

Byte Array	Meaning
{0x01,0x00,0x00,0x00} 0x01000000	Client is authorized to play back the content. This is known as RIGHT_PLAY_ON_PC.
{0x02,0x00,0x00,0x00} 0x02000000	Client is authorized to copy the licensed content to a device that is not compliant with the Secure Digital Music Initiative (for more information, see [SDMI]). This is known as RIGHT_COPY_TO_NONSDMI_DEVICE.
{0x04,0x00,0x00,0x00} 0x04000000	Client is not authorized to restore the license content. This is known as RIGHT_NO_RESTORE.
{0x08,0x00,0x00,0x00} 0x08000000	Client is authorized to burn the licensed content to a CD. This is known as RIGHT_BURN_TO_CD.
{0x10,0x00,0x00,0x00} 0x10000000	Client is authorized to copy the licensed content to a Secure Digital Music Initiative (SDMI) device (for more information, see [SDMI]). This is known as RIGHT_COPY_TO_SDMI_DEVICE.
{0x20,0x00,0x00,0x00}	Client can perform any of the authorized actions once. This is known as

Byte Array	Meaning
0x20000000	RIGHT_ONE_TIME.
{0x00,0x00,0x01,0x00} 0x00000100	Client is authorized to handle SDMI-generated events (for more information, see [SDMI]). This is known as RIGHT_SDMI_TRIGGER.
{0x00,0x00,0x02,0x00} 0x00000200	Client is not authorized to make any further SDMI copies of the licensed content. This is known as RIGHT_SDMI_NOMORECOPIES.

appSec: A 4-byte buffer that contains the application security level submitted by the client.

expiryDate: A 4-byte buffer that contains the date on which the license expires. All values are encoded as hexadecimal. The first byte contains the value of the first two digits of the year, the second contains the value of the last two digits of the year, the third contains the value of the month, and the fourth contains the value of the day. For example, the date 12/30/2002 is represented as {0x14, 0x02, 0x0C, 0x1E}.

Cryptographic Sequence:

$key = \{ K_{content} \mid P_{content} \}_{KM}$

When content is encrypted, the packager generates a content key identifier (KID) and a content key as a pair, using an optional seed value passed to the Microsoft Windows Media Rights Manager SDK. The key is used to encrypt the content and is then discarded, while the KID is placed in the content header of a license request. If used, the seed value used by the server SHOULD be a cryptographically-random bit string. If the seed value is associated with the KID and persisted to some form of long-term storage (for example, a database), the license may be regenerated successfully at a later date if the client has a need to reacquire the same license.

The server can verify the license request's signature to ensure that it has not been tampered with by an external agent. The KID and optional seed value are passed to the Windows Media Rights Manager SDK and the content encryption key is regenerated, after which it is embedded in the license as the key member of the **LICENSEDATA** structure.

The Digital Rights Management (DRM) component on the client computer can use this key to decrypt the content.

2.4 DRM Version 7 Data Types

The following structures and algorithm are specific to version 7 of the Digital Rights Management License Acquisition Protocol.

2.4.1 DRM Version 7 License Request

The Digital Rights Management (DRM) version 7 License Request packet is used by the client to request a license for Windows Media content.

Silent and Nonsilent Requests

The DRM version 7 client can generate a "silent" or a "nonsilent" license request. By contrast, version 1 clients always generate a "nonsilent" license request. Silent license acquisition means that the client application SHOULD NOT display a license acquisition user interface, which requires active user input, to the end user. A client application MAY display some form of progress indicator. Conversely, nonsilent license acquisition means that the client application MAY display a license acquisition user interface, which requires active user input, to the end user.

Silent Requests

Silent acquisition is transparent to the user. The client prefixes the string "nonsilent=0&challenge=", or alternatively just "challenge=", to the head of the encoded data before obtaining the final POST data. Digital Rights Management sends the request directly and receives the response directly. The license is delivered and stored without any other action being required.

Nonsilent Requests

Nonsilent acquisition means that some form of interaction is required from the user. For instance, a Web page might be displayed, requiring the user to enter information, such as a password or an e-mail address. In this case, the POST data is handed back to the application, which prefixes the string "nonsilent=1&challenge=" to the head of the encoded packet, and then makes the HTTP POST.

HTTP POST Headers

nonsilent: Optional specification for silent versus nonsilent license acquisition. If not present, silent license is assumed. Allowable values are "0" to indicate silent license acquisition and "1" to indicate nonsilent license acquisition.

challenge: Required value containing the version 7 license request body.

XML Schema for Version 7 License Request

The following is an XML schema for the version 7 license request packet. Where required, elements, attributes, and values are described in greater detail after the schema.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="LICENSEREQUEST">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="V1CHALLENGE">
          <xs:simpleType>
            <xs:restriction base="xs:base64Binary" />
          </xs:simpleType>
        </xs:element>
        <xs:element name="ACTIONLIST" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="ACTION" type="ActionNameType" minOccurs="1"
                  maxOccurs="5" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="CLIENTINFO" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="CLIENTID" type="xs:base64Binary" />
              <xs:element name="CLIENTVERSION" type="xs:string" />
              <xs:element name="SECURITYVERSION" type="xs:string" />
              <xs:element name="APPSECURITY" type="xs:string" />
              <xs:element name="SUBJECTID1" type="xs:integer" />
              <xs:element name="SUBJECTID2" type="xs:integer" />
              <!-- SUBJECTID2 tag must be present; content is optional. -->
              <xs:element name="DRMKVERSION" type="xs:string"
minOccurs="0" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="WRMHEADER" minOccurs="0" >
```

```
          <xs:complexType>
            <xs:sequence>
              <!-- content varies, depending on media file header
information. -->
              <xs:any />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="version" use="required" fixed="2.0.0.0" />
    </xs:complexType>
  </xs:element>
  <xs:simpleType name="ActionNameType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Play" />
      <xs:enumeration value="Print.redbook" />
      <xs:enumeration value="CREATE_PM_LICENSE" />
      <xs:enumeration value="Backup" />
      <xs:enumeration value="Restore" />
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

2.4.1.1 ACTION

The <ACTION> element contains the action rights that the client is requesting. The meaning of the element contents is described below.

Predefined string	Meaning
Play	Play content on the client computer.
Print.redbook	Burn content to a CD.
CREATE_PM_LICENSE	Transfer content to a portable device.
Backup	Permit backup of the license.
Restore	Allow the license to be restored from another location.

2.4.1.2 APPSECURITY

The <APPSECURITY> element contains the security level of the application making the license request. This value is not limited to a specific range and can be used by the service provider to limit license distribution.

2.4.1.3 CLIENTID (Element)

The <CLIENTID> element contains base64-encoded **CLIENTID** structure.

2.4.1.4 CLIENTID (Structure)

The **CLIENTID** structure contains the Digital Rights Management version and security certificate of the client computer.

```
typedef struct {
```

```
  BYTE Version[4];
  BYTE EncRandNum[80];
  PKCERT pkcert[84];
} CLIENTID;
```

Version: The Digital Rights Management version. MUST be {0x02, 0x00, 0x00, 0x00}.

EncRandNum: One-time use random number encrypted using ECC_1 with KS. The first 7 bytes (unencrypted) of EncRandNum are used as the initialization vector (IV) to create an RC4 key (KR).

pkcert: A **PKCERT** structure that contains the machine certificate.

Cryptographic Sequence:

1. pkcert.pk = KM_{pub}
2. pkcert.sign = $[pkcert.pk]_{KC}$
3. $\{EncRandNum\}_{KS}$
4. KR {pkcert}

2.4.1.5 CLIENTVERSION

The <CLIENTVERSION> element contains the version of the Digital Rights Management client making the request. Generally, this will be of the form "2.a.0.b", where "a" is the minor version and "b" is the client build number.

2.4.1.6 DRMKVERSION

The <DRMKVERSION> element contains the version of the kernel mode Digital Rights Management file (Drmk.sys) on the client computer. Generally, this takes the form "a.b.c.d" where a, b, c, and d are whole numbers.

2.4.1.7 SECURITYVERSION

The <SECURITYVERSION> element contains the security version of the Digital Rights Management root of trust on the client computer.

Each license server has a list of client verification keys that enable it to ensure the validity of license requests. Each verification key string is a base64-encoded **PUBKEY** structure. The list of possible security versions and verification keys is a separate licensable piece of data.

2.4.1.8 SUBJECTID1

The <SUBJECTID1> element contains the certificate subject identifier of the component that is communicating directly with the Digital Rights Management client component. The subject identifier is taken from the certificates that are used to establish secure channels between components. It uniquely identifies a component. For example, the version of an application or an SDK can be a subject identifier.

2.4.1.9 SUBJECTID2

The <SUBJECTID2> element contains the certificate subject identifier of the component that is communicating with another component, which is in turn communicating with the Digital Rights Management client component. This element is used only if <SUBJECTID1> is the subject identifier of an SDK. The subject identifier is taken from the certificates that are used to establish secure channels between components. It uniquely identifies a component. For example, the version of an application or an SDK can be a subject identifier.

The <SUBJECTID2> element MUST be present, but it MAY be empty.

2.4.1.10 V1CHALLENGE

The <V1CHALLENGE> element is a base64-encoded Digital Rights Management version 1 license request created with an empty **KeyID** field.

2.4.1.11 WMRMHEADER

The <WMRMHEADER> element contains data that is taken verbatim from the header of the content. The content is dictated by the Windows Media Rights Manager (WMRM).

For more information about the <WMRMHEADER> and how it is generated, see [MSDN-WMRMHEADOBJ].

2.4.2 DRM Version 7 License Response

The Digital Rights Management (DRM) Version 7 License Response packet is used by the license server to send a license for Windows Media content to a client. The format of the response, which is in extensible markup language (XML), can include any number and combination of Digital Rights Management License Acquisition Protocol version 1 and version 7 licenses, encoded with the base64-encoding algorithm.

Silent Acquisition:

The license response is returned directly to the client as the body of the HTTP response.

Nonsilent Acquisition:

The license response is returned to the client as an HTML page which uses a COM object to store a base64-encoded <LICENSERESPONSE> XML blob in the client's local license store. The <LICENSERESPONSE> XML is embedded in a script section of the Web page.

The normal sequence of operations to store the license locally is as follows:

1. Create an IWMDRMProvider instance via WMDRMCreateProvider or WMDRMCreateProtectedProvider.
2. Create an IWMDRMLicenseManagement instance by calling IWMDRMProvider::CreateInstance, passing in the IID of IWMDRMLicenseManagement.
3. Call IWMDRMLicenseManagement::StoreLicense to store the license locally.

For more information about these interfaces and functions, see [MSDN-WMF11SDK].

Errors

If the license is refused after a nonsilent request, the license server responds with an error code in the browser window. If the license is refused after a silent request, the server returns one of the following error codes as the HTTP status code.

Error	Value	Meaning
NS_E_DRM_LICENSE_NOTACQUIRED	0xC00D2759	Error occurred while acquiring the license.
NS_E_DRM_LICENSE_STORE_ERROR	0xC00D2712	Error occurred while storing the license.

If the client application receives one of these errors, it attempts a nonsilent request.

XML Schema for Version 7 License Response:

The following is an XML schema for the Version 7 license response packet per [XML], [XMLSCHEMA1/2], and [XMLSCHEMA2/2]. Where required, elements, attributes, and values are described in greater detail after the schema. Note that this is not a strictly correct XML schema, as the <LICENSE> element may be either a version 1 license or a version 7 license. The "version" attribute of the <LICENSE> element differentiates the two.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="LICENSERESPONSE">
    <xs:complexType>
      <xs:choice>
        <xs:element name="REVOCATION" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:base64Binary">
                <xs:attribute name="type" use="required" type="RevocationType" />
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
        <xs:element name="REVOCATIONINFO" minOccurs="0">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:base64Binary"/>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:choice>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="LICENSE">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:base64Binary">
                <xs:attribute name="version" use="required" fixed="0.1.0.0" />
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
        <xs:element name="LICENSE">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:base64Binary">
                <xs:attribute name="version" use="required" fixed="2.0.0.0" />
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:choice>
```

```
      </xs:complexType>
    </xs:element>
    <xs:simpleType name="RevocationType">
      <xs:restriction base="xs:string">
        <xs:enumeration value="WMDRMNET" />
        <xs:enumeration value="DeviceRevocationList" />
        <xs:enumeration value="RevocationList" />
        <xs:enumeration value="{66DD5134-4E34-40ae-9D5D-13A112B7591F}" />
      </xs:restriction>
    </xs:simpleType>
  </xs:schema>
```

2.4.2.1 REVOCATION

The <REVOCATION> element contains a base64-encoded REV_INFO structure. For more information about the REV_INFO structure, see [MSDN-WMF11SDK].

2.4.2.2 REVOCATIONINFO

The <REVOCATIONINFO> element contains a base64-encoded REV_INFO structure. For more information about the REV_INFO structure, see [MSDN-WMF11SDK].

2.4.3 DRM Version 1 License Format

The version 1 <LICENSE> element contains a base64-encoded version 1 CERTIFIED_LICENSE.

2.4.4 DRM Version 7 License Format

A Digital Rights Management License Acquisition Protocol version 7 license is represented in extensible markup language (XML) format. The version 7 <LICENSE> element contains a base64-encoded XML string representing the version 7 license. The schema for a version 7 license is given below.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="LICENSE">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="LICENSORINFO">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="DATA">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="LID" type="xs:string" />
                    <xs:element name="KEYID" type="xs:string" />
                    <xs:element name="ISSUEDATE" type="xs:date" />
                    <xs:element name="CONTENTPUBKEY" type="xs:base64Binary" />
                    <xs:element name="PRIORITY" type="xs:integer" />
                    <xs:element name="META" minOccurs="0">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:any />
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="ONSTORE"
                      <xs:complexType>
```

```
                        <xs:sequence>
                          <xs:element name="CONDITION" type="xs:string"
                              minOccurs="0" />
                          <xs:element name="ACTION" type="xs:string"
                              minOccurs="0" />
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="ONSELECT">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="CONDITION" type="xs:string" />
                          <xs:element name="ACTION" type="xs:string"
minOccurs="0" />
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="ONCLOCKROLLBACK" minOccurs="0">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="ACTION" type="xs:string"
minOccurs="0" />
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="ONACTION" minOccurs="0">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="CONDITION" type="xs:string" />
                          <xs:element name="ACTION" type="xs:string"
minOccurs="0" />
                        </xs:sequence>
                        <xs:attribute name="type" type="ActionNameType" />
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="ENABLINGBITS">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="ALGORITHM" minOccurs="0">
                            <xs:complexType>
                              <xs:sequence />
                              <xs:attribute name="type" use="required"
                                  fixed="MSDRM" />
                            </xs:complexType>
                          </xs:element>
                          <xs:element name="PUBKEY">
                            <xs:complexType>
                              <xs:simpleContent>
                                <xs:extension base="xs:base64Binary">
                                  <xs:attribute name="type" use="required"
                                      fixed="machine" />
                                </xs:extension>
                              </xs:simpleContent>
                            </xs:complexType>
                          </xs:element>
                          <xs:element name="VALUE" type="xs:base64Binary" />
                          <xs:element name="SIGNATURE"
type="xs:base64Binary" />
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="CONTENTREVOCATION" minOccurs="0"
                        maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:sequence>
```

```
                        <xs:element name="DATA">
                          <xs:complexType>
                            <xs:sequence>
                              <xs:element name="SEQUENCENUMBER"
                                  type="xs:integer" />
                              <xs:element name="CONTENTPUBKEY"
                                  type="xs:base64Binary" />
                              <xs:element name="LICENSESERVERPUBKEY"
                                  type="xs:base64Binary" />
                              <xs:element name="CONDITION"
type="xs:string" />
                            </xs:sequence>
                          </xs:complexType>
                        </xs:element>
                        <xs:element name="SIGNATURE">
                          <xs:complexType>
                            <xs:sequence>
                              <xs:element name="HASHALGORITHM">
                                <xs:complexType>
                                  <xs:sequence />
                                  <xs:attribute name="type" fixed="SHA"
                                      use="required" />
                                </xs:complexType>
                              </xs:element>
                              <xs:element name="SIGNALALGORITHM" minOccurs="0">
                                <xs:complexType>
                                  <xs:sequence />
                                  <xs:attribute name="type" fixed="MSDRM" />
                                </xs:complexType>
                              </xs:element>
                              <xs:element name="VALUE" type="xs:base64Binary" />
                            </xs:sequence>
                          </xs:complexType>
                        </xs:element>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="SIGNATURE">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="HASHALGORITHM">
                    <xs:complexType>
                      <xs:sequence />
                      <xs:attribute name="type" fixed="SHA"
use="required" />
                    </xs:complexType>
                  </xs:element>
                  <xs:element name="SIGNALALGORITHM" minOccurs="0">
                    <xs:complexType>
                      <xs:sequence />
                      <xs:attribute name="type" fixed="MSDRM" />
                    </xs:complexType>
                  </xs:element>
                  <xs:element name="VALUE" type="xs:base64Binary" />
                </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="CERTIFICATECHAIN">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="CERTIFICATE" type="xs:base64Binary"
                      minOccurs="2" maxOccurs="unbounded" />
```

```
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="version" use="required" fixed="2.0.0.0" />
  </xs:complexType>
 </xs:element>
 <xs:simpleType name="ActionNameType">
  <xs:restriction base="xs:string">
   <xs:enumeration value="Play" />
   <xs:enumeration value="Print.redbook" />
   <xs:enumeration value="CREATE_PM_LICENSE" />
   <xs:enumeration value="Backup" />
   <xs:enumeration value="Restore" />
  </xs:restriction>
 </xs:simpleType>
</xs:schema>
```

For a sample version 7 license, see DRM Version 7 License Example.

2.4.4.1 ACTION

The <ACTION> element contains a single expression that defines the action to take after an event is raised.

The return value of the expression is ignored and may be of any type.

The action to take is contained in a **CDATA** block, as in the following example.

```
<ACTION>
 <![CDATA[
   deletelicense()
 <]]>
</ACTION>
```

2.4.4.2 CERTIFICATE

The <CERTIFICATE> element contains a base64-encoded version 1 **CERT** structure.

2.4.4.3 CERTIFICATECHAIN

The <CERTIFICATECHAIN> element specifies the certificate chain, which contains the credentials needed to issue licenses. This element is used to inform the Digital Rights Management client that the license server is authorized to issue licenses.

The first <CERTIFICATE> child element is the certificate issued by the root authority (CR). The second <CERTIFICATE> child element is the license server certificate (CS). Subsequent <CERTIFICATE> child elements comprise a certificate chain downward from CS.

2.4.4.4 CONDITION (ONACTION, ONSELECT, ONSTORE)

The <CONDITION (ONACTION, ONSELECT, ONSTORE)> element is an expression that is evaluated to determine if an event is allowed. This evaluation is done before the event is performed.

A condition is an expression. The return value of the expression must be an integer (**LONG**).

A condition that evaluates to 0 is **FALSE**; one that evaluates to non-zero is **TRUE**. If the condition is **TRUE**, the event is allowed; otherwise, it is not.

The condition is contained in a **CDATA** block, as in the following example.

```
<CONDITION>
  <![CDATA[
    secstate.playcount > 0
   ]]>
</CONDITION>
```

2.4.4.5 CONDITION (CONTENTREVOCATION/DATA)

The <CONDITION (CONTENTREVOCATION/DATA)> element contains an expression that describes the condition under which the content owner selects licenses to be revoked. The default condition element that the server generates specifies that a particular license should be deleted, and no other events allowed. No other conditions are generated.

The default condition element is as follows:

```
<CONDITION>
  <![CDATA[ deletelicense();0 ]]>
</CONDITION>
```

2.4.4.6 CONTENTPUBKEY

The <CONTENTPUBKEY> element contains the base64-encoded public key of the content packager (KS). It is used to verify the signature in the content header.

When it is part of the <CONTENTREVOCATION> string, it is used to identify an existing license on a client computer. The revocation string revokes all licenses that have this <CONTENTPUBKEY>.

2.4.4.7 CONTENTREVOCATION

The <CONTENTREVOCATION> element enables a content owner to disable licenses issued by that owner.

2.4.4.8 ENABLINGBITS

The <ENABLINGBITS> element contains the encrypted key and information needed to unlock the content.

2.4.4.9 Events in DRM Licenses

An event is an element of a license. Each event has a <CONDITION (ONACTION)> expression to be evaluated and an optional <ACTION> expression. A condition that evaluates to 0 is false; one that evaluates to nonzero is true. If the condition is true, the event is allowed and the action is taken.

A license can specify any of the following events.

Event	Description
<ONSTORE>	Raised when the license is stored.
<ONSELECT>	Raised when the license is selected.
<ONCLOCKROLLBACK>	Raised when the client detects a clock rollback.
<ONACTION>	Raised when the application queries or consumes a requested right.

2.4.4.10 Expressions in DRM Licenses

An expression in a Digital Rights Management (DRM) license is a combination of operators and identifiers that specifies a computation of a value, or designates a variable or a constant.

The <CONDITION (ONACTION, ONSELECT, ONSTORE)> and <ACTION> elements of a license are expressions. An expression can consist of the following items.

2.4.4.10.1 Identifier

An identifier is a sequence of characters that starts with an alphanumeric character and consists entirely of any number and combination of alphabetical characters, digits, underscores ("_"), and the dot (".") symbol. The characters are case-sensitive.

2.4.4.10.2 Function Symbol

An identifier is a function symbol if it is included in the list of predefined functions in DRM expressions.

2.4.4.10.3 Constants

A constant is an identifier of the **DATETIME**, **LONG**, or **STRING** data type, as specified in section 2.4.4.12.

2.4.4.10.4 Variable

An identifier is a variable if it is not a function symbol or a constant. Variables can be valid or invalid. A valid variable starts with one of the predefined prefix categories, which are specified in section 2.4.4.23. It is followed by a dot symbol and an attribute (for example, machine.datetime or app.count).

An invalid variable is one that is not in the list of prefixes. If a variable is not valid, the expression evaluation terminates and the expression is treated as false.

The value of a variable is retrieved from a specific location, which depends on the variable's category. For example, content.CID is retrieved from the content header, and license.LID is retrieved from the license.

The existence of a variable can be checked with the **exists** function.

2.4.4.10.5 Final Value

The final value of an expression is one of the three data types allowed in expressions: **DATETIME**, **LONG**, or **STRING**.

A <CONDITION (ONACTION)> expression must result in a **LONG**. If the final value is 0, the condition is considered false; if the final value is non-zero, the condition is considered true.

The result of an <ACTION> expression can be of any type because the final value is ignored.

2.4.4.11 Operators in DRM Expressions

A Digital Rights Management license expression can include some of the operators that are found in the C programming language. All binary operators are used in infix notational form.

2.4.4.11.1 Operator Behavior

The behavior of operators in an expression depends on the types of the operands. The following table lists the allowed operators and the results they produce with operands of various types.

Operator	Operand1	Operand2	Result description
+	**LONG**		Unary plus.
+	**LONG**	**LONG**	Binary addition.
+	**STRING**	**STRING**	Concatenation of strings.
-	**LONG**		Unary minus.
-	**LONG**	**LONG**	Binary subtraction.
*	**LONG**	**LONG**	Binary multiplication.
/	**LONG**	**LONG**	Integer division (for example, 7/3 = 2).
%	**LONG**	**LONG**	Modulo operator (for example, 7 % 3 = 1).
++	**LONG**		Unary post-increment or pre-increment operator. Variable should support set operations.
--	**LONG**		Unary post-decrement or pre-decrement operator. Variable should support set operations.
=	**LONG**	**LONG**	Simple assignment.
=	**STRING**	**STRING**	Simple assignment.
=	**DATETIME**	**DATETIME**	Simple assignment.
< <= > >= == !=	**LONG**	**LONG**	Relational operator. Result is a **LONG** with a value of 0 or 1.
< <= > >= == !=	**STRING**	**STRING**	Relational operator. Result is a **LONG** with a value of 0 or 1.
< <= > >= == !=	**DATETIME**	**DATETIME**	Relational operator. Result is a **LONG** with a value of 0 or 1.
!	**LONG**		Unary Not. Result is a **LONG** with a value of 0 or 1.

Operator	Operand1	Operand2	Result description
&&	**LONG**	**LONG**	Logical AND. Result is a **LONG** with a value of 0 or 1. Shortcut evaluation is supported. For example, in the expression "a && b", if "a" is false, "b" is not evaluated.
||	**LONG**	**LONG**	Logical OR. Result is a **LONG** with a value of 0 or 1. Shortcut evaluation is supported. For example, in the expression "a || b", if "a" is true, "b" is not evaluated.
;	**LONG**	**LONG**	Separates two expressions. Each is evaluated in turn, but the result is the value of the second operand. For example, in the expression "1+2;4", the operand "1+2" is evaluated, but the resulting value is "4".
()			Allows precedence to be overridden.
?:	Any	Any	Conditional expression; for example, "(a < b)?c:d". If condition "a < b" is true, the value is "c", and "d" is not evaluated. If the condition "a < b" is false, the value is "d", and "c" is not evaluated.
,	Any	Any	Used to separate parameters in a function call or used in an expression to allow multiple statements to be evaluated. For example, "d = (a = b, c = e)" will assign the value of "e" to "d".

2.4.4.11.2 Operator Precedence

The following list shows the precedence of the operators in the table above, from highest to lowest.

( )
FunctionCall
! ++ -- +(unary) -(unary)
* / %
+ -
< > <= =>
== !=
&&
||
?:
=
;
,

2.4.4.12 Data Types in DRM Expressions

A Digital Rights Management expression, and the constants used in it, can have one of three data types: **DATETIME**, **LONG**, or **STRING**.

2.4.4.12.1 DATETIME

This data type is represented by a specific syntax, which takes one of the following formats:

#YYYYMMDDZ #

#YYYYMMDD HH:MM:SSZ#

The time is represented in Coordinated Universal Time (UTC) format. This portion is optional, and if it is missing, it is assumed to be zero. The Z at the end of the line indicates that the time is in UTC and is required even if the time portion is not present.

2.4.4.12.2 LONG

This data type is represented by an integer, which can be either decimal or hexadecimal. Hexadecimal values must be prefixed by the string "0x". This type is also used to represent Boolean values, where 0 is false and nonzero is true.

2.4.4.12.3 STRING

This data type is represented by double quotation marks (""). A **STRING** can include any character except the double quotation marks. However, a double quotation mark can be represented by using a backslash ("\"). For example, the string ""ab\"cd"" is rendered as "ab"cd".

The backslash, also referred to as the escape character, can represent a newline character when it is placed before the letter n, as in "\n". The escape character itself can be represented with two backslashes ("\\"). If the escape character is followed by any character other than n, ", or \, the pair of characters is replaced with the character that follows the backslash. For example, "\a" is equivalent to "a".

2.4.4.12.4 Casting Data Types

Implicit casting is not allowed. For example, the plus sign ("+") cannot be applied to **DATETIME** operands. However, the **DATETIME**, **LONG**, and **STRING** types can be used as casting operators. The following table lists the possible type conversions.

Conversion	Allowed
DATETIME to **LONG**	No
DATETIME to **STRING**	Yes
LONG to **DATETIME**	No
LONG to **STRING**	Yes
STRING to **LONG**	Yes
STRING to **DATETIME**	Yes

2.4.4.13 KEYID

The <KEYID> element contains the identifier of the key associated with a license.

The <KEYID> MAY be any text and SHOULD NOT be binary. Use of a base64-encoded **globally unique identifier (GUID)** is recommended.

2.4.4.14 LICENSESERVERPUBKEY

The <LICENSESERVERPUBKEY> element contains the public key of the license server (KS).

2.4.4.15 LICENSORINFO

The <LICENSORINFO> element contains license details provided by the license issuer.

2.4.4.16 LID

The <LID> element is a unique license identifier that is automatically created by the license generator. It must be a **curly braced GUID string**, as shown in the following example.

Example Code

```
<LID>{00000507-0000-0010-8000-00AA006D2EA4}</LID>
```

2.4.4.17 META

The use of this element is dependent on the client application. The child elements of <META> are optional and can contain metadata about the license. Digital Rights Management does not depend on specific fields.

2.4.4.18 ONACTION

The <ONACTION> element is an event that is raised when the application consumes or queries a specific action right.

An <ONACTION> event is required for each right that the license allows. If the event is missing, the right is not allowed.

The condition associated with this event is evaluated when the application consumes or queries the requested right. If the condition is true, the action is allowed; if the condition is false, the action is not allowed. If the condition is missing, it is assumed to be true.

If this event has an <ACTION> element, the action is taken after the right is consumed. If the application queries only the right, the action is not taken. The action's final value is ignored.

2.4.4.19 ONCLOCKROLLBACK

The <ONCLOCKROLLBACK> element is an event that is raised when the Digital Rights Management (DRM) client detects a clock rollback. It gives the license a means of reacting to the rollback.

This event has an action only; there are no conditions to evaluate.

When the DRM client detects a clock rollback, it gives every license that includes this event a chance to react to it. The license must specify the action to take; DRM simply evaluates the expression in the action.

For example, upon detecting clock rollback, a license may indicate that it must be deleted.

2.4.4.20 ONSELECT

The <ONSELECT> element is an event that is raised when the license is selected.

The condition for this event is evaluated when the license is selected. If the condition is true, the license can be selected; if it is false, the license cannot be selected. If the condition is missing, it is assumed to be true.

If this event has an <ACTION> element, the action is evaluated after the license is selected. The action's final value is ignored.

If the license does not include this event, no conditions for license selection are present, and the license is selected.

For more information about allowable conditions, see section 2.4.4.9.

2.4.4.21 ONSTORE

The <ONSTORE> element is an event that is raised when the license is stored.

The condition for this event is evaluated when the license is stored. If it is true, the license is stored; if it is false, the license is not stored. If the condition is missing, it is assumed to be true.

If this event has an <ACTION> element, the action is taken after the license is stored.

If the license does not include this event, no conditions for license storage are present, and the license is stored. The action's final value is ignored.

2.4.4.22 Predefined Functions in DRM Expressions

An event is an element of a license. Events can contain predefined function calls. Functions are evaluated as soon as the argument list is closed.

The following table lists and describes the supported functions.

Function	Arg1	Arg2	Arg3	Description
min	LONG	LONG		Returns the smaller of the two arguments. The result is **LONG**.
max	LONG	LONG		Returns the larger of the two arguments. The result is **LONG**.
long	STRING			Converts **STRING** to **LONG**. **STRING** has the syntax "[whitespace][sign][number]". The **number** attribute should have at least one digit, which can be decimal or hexadecimal. No white space is allowed after **sign**, but trailing spaces are allowed.
long	LONG			Performs an **identity** operation.
string	LONG			Converts **LONG** to **STRING**.
string	STRING			Performs a string **identity** operation.
string	DATETIME			Converts **DATE** to **STRING**.
datetime	STRING			Converts **STRING** to **DATE**.

Function	Arg1	Arg2	Arg3	Description
datetime	DATETIME			Performs an **identity** operation for **DATE**.
dateadd	STRING	LONG	DATETIME	Adds date elements. Arg1 can be "d" (days), "h" (hours), "n" (minutes), or "s" (seconds). The corresponding amount specified in Arg2 is added to the given **datetime** to get the target date and time. The result is a **DATETIME**.
datediff	STRING	DATETIME	DATETIME	Subtracts Arg2 from Arg3. The result is given in units, as indicated in Arg1. Arg1 can be "d", "h", "n", or "s". The result is a **LONG**.
datepart	STRING	DATETIME		Returns an integer that represents the specified datepart (Arg1) of Arg2.Arg1 can be "y", "m", "d", "h", "n", or "s". The result is a **LONG**.
index	STRING	STRING		Returns the index of Arg1 in Arg2, if it is found. The first index is 0. If it is not found, return -1. The result is a **LONG**.
length	STRING			Returns the length of Arg1. The result is a **LONG**.
deletelicense				Deletes the current license, returning 1 if successful, 0 otherwise.
exists	variable			Determines whether a variable exists. Returns TRUE if successful, FALSE otherwise.
versioncompare	string	string		Compares two strings, treating them as versions. If they are not versions, the result is undefined. A version string has the form "<n>.<n>.<n>.<n>", where "<n>" is a number.

2.4.4.23 Predefined Variables in DRM Expressions

Each license must create and access its own unique collection of attribute/value pairs. After being created, they cannot be deleted. A license can access only the attribute/value pairs that it created.

The attributes must belong to one of the following categories:

- drm
- drmk
- license
- pmlicense
- content

- machine
- server
- app (application)
- secstate (secure state in the client)

2.4.4.23.1 Attributes

The following table enumerates all possible attributes that a license can expose.

Variable	Data type	Description
drm.version	**STRING**	The Digital Rights Management (DRM) version. This variable does not use the build number, but rather the hard-coded value in the client.
drm.bb.msdrm.version	**STRING**	The current DRM root of trust version. This data is not necessarily secure.
drmk.version	**STRING**	The version of the kernel mode DRM file (DRMK) on the client computer. This variable does not exist on a computer without DRMK. Use **exists** (drmk.version) to check for the presence of DRMK.
drmk.parameter	**STRING**	A string to use to set up DRMK. The string should take the form "attr=value;attr=value;" and so on. Supported attributes are **spdif**, **certs**, and **mindrmdriverlevel**. The default values for all are true, false, and 1000. If **certs** is true, certified drivers are required. The **mindrmlevel** attribute indicates the level of security needed for the drivers; do not use it if **certs** is false. The **spdif** attribute allows the transfer of audio from one file to another without conversion to and from an analog format. If true, this type of transfer is allowed.
machine.datetime	**DATETIME**	The time in Coordinated Universal Time (UTC) format, based on the client computer's clock.
app.count	**LONG**	The number of DRM certificates used currently. The value is 2 if an application uses the WMRM SDK, and 1 if an application uses DRM directly.
app.minseclevel	**LONG**	The minimum security level, computed from the supplied application certificates.
secstate.<attribute>	Any	The specified attribute value (for example, "secstate.firstdateofuse"). If the attribute does not exist, an error is returned. For assignments, the attribute is created if it does not already exist. Its type is the same as the type of the value assigned to it.
secstate.global.saveddatetime	**DATETIME**	The last saved clock time, as recorded by the DRM system. This is particularly useful for the <ONCLOCKROLLBACK> event. It is a read-only field for the license.
license.<attribute>	**STRING**	The value of the attribute in the license <LICENSORINFO>/<DATA> section (for example,

Variable	Data type	Description
		"license.LID" or "license.KID"). The attribute is case-sensitive. It is possible that the value is an extensible markup language (XML) string. For example, license.META gives the entire XML string for the <META> section, without the META tags.
content.<attribute>	**STRING**	The value of the attribute in the content header DATA section (for example, "content.CID").
pmlicense.version (see note below)	**STRING**	The version of the Portable Media (PM) license being requested. This field is read-only and can be used in the <CONDITION (ONACTION, ONSELECT, ONSTORE)> section of the rights that provide the PM license.
pmlicense.rights (see note below)	**LONG**	The rights to use for generating the PM license, if creating one is allowed. The default value is 0.
pmlicense.appseclevel (see note below)	**LONG**	The application security level to use for generating the PM license, if creating one is allowed. The default value is 0.
pmlicense.expirydate (see note below)	**DATETIME**	The date to use for generating the PM license, if creating one is allowed. The default value is #19991231Z#.

Note A license server can issue a Portable Media license, which supports the moving of content to devices other than a PC. A server can automatically include this right when issuing a license for a request in which Play is the only requested right. The license is based on the DRM Version 1 License Format. In the DRM Version 7 License Format, the last four variables in the table above are used in the <ONACTION> element to provide the means of generating a PM license.

2.4.4.24 PRIORITY

The <PRIORITY> element indicates the priority of the license, helping the client choose the appropriate one when multiple licenses are available for the same content. A license with a higher priority is consumed before one with a lower priority.

The value of this element is an integer from 0 to 2147483647. The license enumeration process on the client selects licenses based on this priority.

2.4.4.25 PUBKEY

The <PUBKEY> element contains a base64-encoded public key to which the enabling bits are bound.

Example code:

```
<PUBKEY type="machine">WEJKJKJKert==</PUBKEY>
```

2.4.4.26 SEQUENCENUMBER

The <SEQUENCENUMBER> element contains the sequence number of the content revocation. This number, which must be an integer, is generated sequentially by the content owner. It is used to determine whether the current content revocation should override an existing one. Content revocation information with higher sequence numbers overrides content revocation information with lower sequence numbers.

2.4.4.27 SIGNATURE (CONTENTREVOCATION, LICENSORINFO)

The <SIGNATURE (CONTENTREVOCATION, LICENSORINFO)> element contains a signature of the <DATA (CONTENTREVOCATION)> or <DATA (LICENSORINFO)> element in the <VALUE> child element.

The key is signed by the license server. This prevents the license from being altered by an external agent.

Cryptographic sequence:

<VALUE> contents = $[\text{<DATA> contents}]_{KS}$

2.4.4.28 SIGNATURE (ENABLINGBITS)

The <SIGNATURE (ENABLINGBITS)> element contains a signature of the <PUBKEY (ENABLINGBITS)> element contents.

Cryptographic sequence:

<SIGNATURE> contents = $[\text{<PUBKEY> contents}]_{KS}$

2.4.4.29 VALUE (ENABLINGBITS)

The <VALUE (ENABLINGBITS)> element contains the base64-encoded, encrypted content decryption key.

The key is encrypted with the computer's public key so that only the client that requested this license can decrypt it.

Cryptographic sequence:

<VALUE> contents = $\{\text{content decryption key}\}_{KM}$

2.5 DRM Version 11 Data Types

The following structures and algorithm are specific to version 11 of the Digital Rights Management License Acquisition Protocol.

2.5.1 DRM Version 11 License Request

The Digital Rights Management (DRM) version 11 license request is almost identical to the version 7 license request. In version 11 license request packets, the <CLIENTINFO> element has two additional child elements, <MACHINECERTIFICATE> and <REVINFO>, described below.

XML Schema for Version 11 License Request

The following is an XML schema for the version 1 license request packet. Where required, elements, attributes, and values are described in greater detail after the schema.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="LICENSEREQUEST">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="V1CHALLENGE">
          <xs:simpleType>
            <xs:restriction base="xs:base64Binary" />
```

```
          </xs:simpleType>
        </xs:element>
        <xs:element name="ACTIONLIST" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="ACTION" type="ActionNameType"
                          minOccurs="1" maxOccurs="10" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="CLIENTINFO" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="CLIENTID" type="xs:base64Binary" />
              <xs:element name="MACHINECERTIFICATE" type="xs:base64Binary" />
              <xs:element name="REVINVO" type="xs:base64Binary" />
              <xs:element name="CLIENTVERSION" type="xs:string" />
              <xs:element name="SECURITYVERSION" type="xs:string" />
              <xs:element name="APPSECURITY" type="xs:string" />
              <xs:element name="SUBJECTID1" type="xs:integer" />
              <xs:element name="SUBJECTID2" type="xs:integer" />
              <!-- SUBJECTID2 tag must be present; content is optional. -->
              <xs:element name="DRMKVERSION" type="xs:string" minOccurs="0" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="WRMHEADER">
          <xs:complexType>
            <xs:sequence>
              <!-- content varies, depending on media file header information. -->
              <xs:any />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="version" use="required" fixed="2.0.0.0" />
    </xs:complexType>
  </xs:element>
  <xs:simpleType name="ActionNameType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Play" />
      <xs:enumeration value="Print.redbook" />
      <xs:enumeration value="CREATE_PM_LICENSE" />
      <xs:enumeration value="Backup" />
      <xs:enumeration value="Restore" />
      <xs:enumeration value="CollaborativePlay" />
      <xs:enumeration value="Copy" />
      <xs:enumeration value="Transfer.SDMI" />
      <xs:enumeration value="Transfer.NONSDMI" />
      <xs:enumeration value="PlaylistBurn" />
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

2.5.1.1 MACHINECERTIFICATE

The <MACHINECERTIFICATE> element contains a base64-encoded copy of the machine certificate (CM).

2.5.1.2 REVINFO

The <REVINFO> element contains a base64-encoded copy of the revocation list known to the client application. If this revocation list is out of date, the license server may provide an updated revocation list in the <CONTENTREVOCATION> element.

2.5.2 DRM Version 11 License Response

The version 11 license response is identical to the version 7 license response.

3 Protocol Details

The following sections specify details of the Digital Rights Management License Acquisition Protocol, including abstract data models, interface method syntax, and message processing rules.

3.1 Client Details

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

3.1.2 Timers

None.

3.1.3 Initialization

None.

3.1.4 Higher-Layer Triggered Events

None.

3.1.5 Client Message Processing Events and Sequencing Rules

3.1.5.1 DRM Version 1 Client Message Processing Events and Sequencing Rules

The Digital Rights Management (DRM) version 1 License Request packet is used by the client to request a license for Windows Media content.

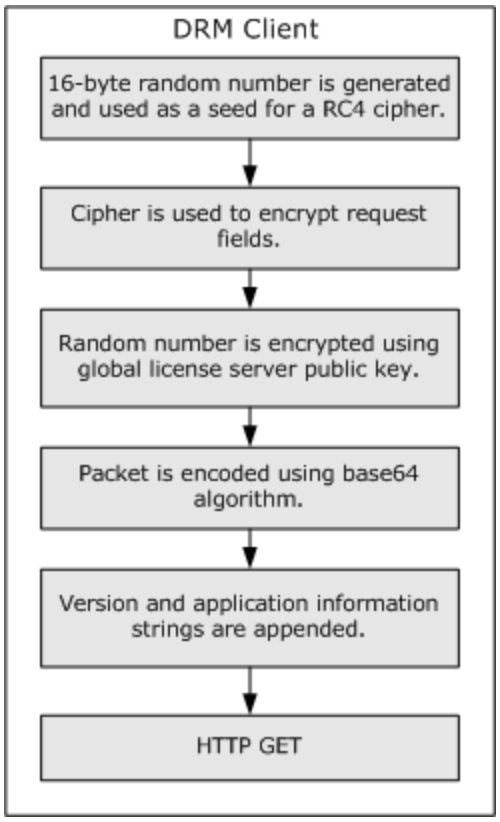

Figure 3: DRM version 1 request packet process

The client generates a license request and sends it to a license server. The request is a binary string that is partially RC4-encrypted and then encoded by using the base64 encoding algorithm.

To create a license request, the client generates a 16-byte random number. This number is used as a seed to create an RC4 cipher, with which some of the request fields are encrypted.

The random number is then encrypted with a global license server public key.

After encryption, the string is base64-encoded. Digital Rights Management then appends the string "&DRMVer=1.4". Note that the value for "DRMVer" may vary based on the client application. The Windows Media Player appends "&embedded=true" if the player is embedded in an HTML page, or "&embedded=false" otherwise. The resulting string is a uniform resource locator (URL) that is sent to the license server as an HTTP GET request.

3.1.5.2 DRM Version 7 Client Message Processing Events and Sequencing Rules

The Digital Rights Management (DRM) version 7 License Request packet is used by the client to request a license for Windows Media content.

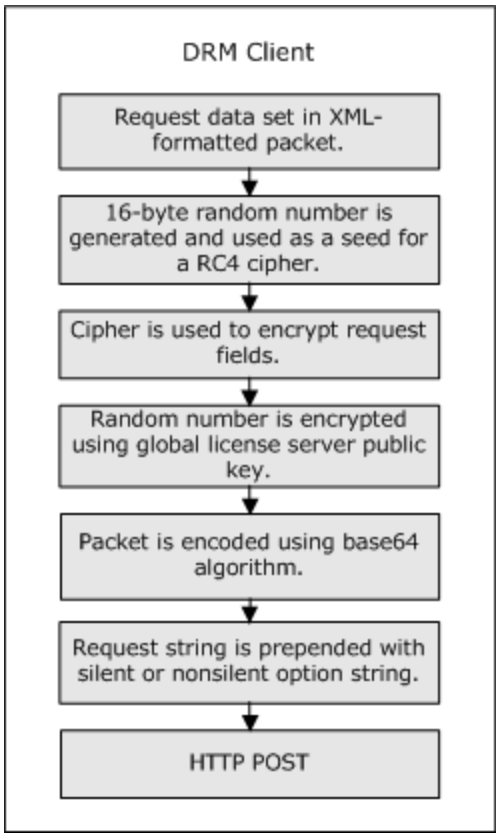

Figure 4: DRM version 7 request packet process

The DRM client generates the license request in XML format and sends it to a license server by using an HTTP POST request. A version 7 license request packet consists of a version 1 challenge, in addition to client and media elements. The client creates a 16-byte random number, which it uses as a seed to create an RC4 cipher. The cipher is used to encrypt some of the request elements, after which the random number is encrypted with a global license server public key. After the encryption process, the packet is encoded by using the base64 encoding algorithm.

3.1.5.3 DRM Version 11 Client Message Processing Events and Sequencing Rules

The Digital Rights Management (DRM) version 1 license request packet is used by the client to request a license for Windows content.

The version 11 processing sequence is identical to the version 7 license response processing sequence.

3.1.6 Timer Events

None.

3.1.7 Other Local Events

None.

3.2 Server Details

3.2.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

3.2.2 Timers

None.

3.2.3 Initialization

None.

3.2.4 Higher-Layer Triggered Events

None.

3.2.5 Server Message Processing Events and Sequencing Rules

3.2.5.1 DRM Version 1 Server Message Processing Events and Sequencing Rules

The Digital Rights Management (DRM) Version 1 License Response packet is used by the license server to send a license for Windows Media content to a client.

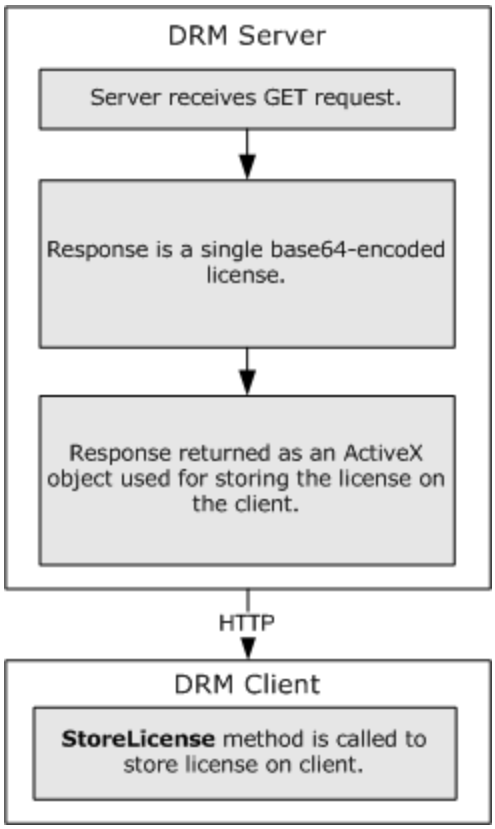

Figure 5: DRM client/server response sequence

The server response to the client's request is a single license encoded with the base64 encoding algorithm. It is returned to the client as part of an HTML page that contains an Internet Explorer ActiveX control or a Netscape plug-in.

The body of the Web page calls the **StoreLicense** method to send the response to the DRM client so that it can store the license.

3.2.5.2 DRM Version 7 Server Message Processing Events and Sequencing Rules

The Digital Rights Management (DRM) version 7 License Response packet is used by the license server to send a license for Windows Media content to a client.

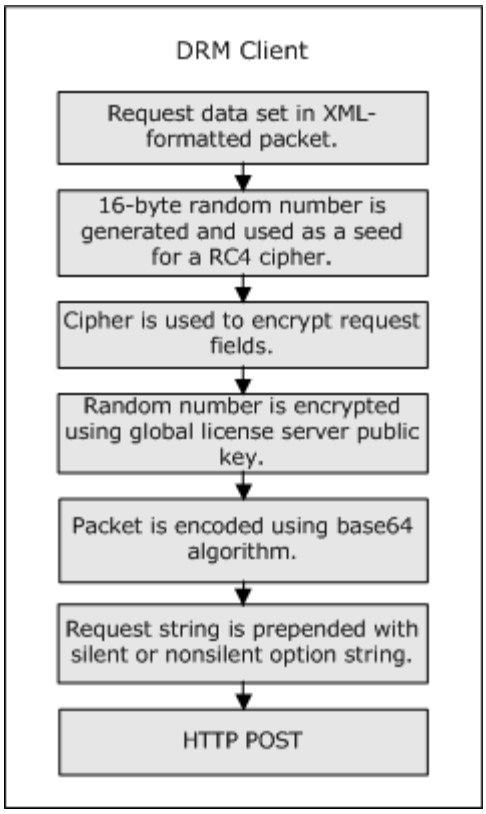

Figure 6: DRM client/server response sequence

If the client request was silent, the response is returned directly to the client. If the request was non-silent, the response is returned to the client as part of an HTML page that contains an Internet Explorer ActiveX control or Netscape plug-in.

The body of the Web page calls the **StoreLicense** method to send the response to the DRM client so that it can store the license.

The format of the response, which is in extensible markup language (XML), can include any number and combination of DRM version 1 and version 7 licenses, encoded with the base64 encoding algorithm.

3.2.5.3 DRM Version 11 Server Message Processing Events and Sequencing Rules

The Digital Rights Management (DRM) Version 11 License Response packet is used by the license server to send a license for Windows Media content to a client.

The version 11 processing sequence is identical to the version 7 license response processing sequence.

3.2.6 Timer Events

None.

3.2.7 Other Local Events

None.

4 Protocol Examples

The following sections describe several operations as used in common scenarios to illustrate the function of the Digital Rights Management License Acquisition Protocol.

4.1 DRM Version 1 License Response Example

The following example shows a response Web page that contains an encoded license.

```
<HTML>
<HEAD>
    <TITLE>V1 Licensing</TITLE>
    <Script Language="VBScript">
        const LICENSE EXPIRED           = &H80041000
        const LICENSE INCONSISTENT      = &H80041001
        const LICENSE INCORRECT VERSION = &H80041003
        Sub Window_OnLoad()
            On Error Resume Next
            DrmStore.StoreLicense(
                "AAEAAHUAAAB3IFadTI8UJy3PzB9yilDoxgf5DRjqL4NXqFkns7*!
                Z4jFwCPX!oCDS1pPTHhMcmhaVStId0dMSlY4V3RhUT09AEeTvPQpG
                Nt!AJ5BE6tB4ZJ5tDQJo*bnTOnAxatFIYch72C8A04kdFz8ZK*!UT
                j52e4dIRkQkMBHXXnma4xe9KFZB3QypiOMM6LQFyPs0ViJGwAAAAA
                AAJYUBwMWAAEAADgAAADRQt0mNlnxj7as*ys3NSMJaaWViZC1Ppnl
                LxYqUdqCMm2iPIlzxu4zm5xxu39qj47qy33j5mXGbpviYTFldxMwN
                RRSckf6kyEdHDya3LyAc2NjDB8AAAAAAAAAAAAAAAAAAQAAOAAAAG
                N!793njE8kEVW*BhFk*W5xfYgP*ymWlfUQely7kQCMci!Q6wPkIhG
                9LfC2Z85Uf01UPGTZ7pNCns0OdMfy85CZ5ceKkC0KYaQK*OrdqAQN
                Y2MMHwAAAGMAAAABAAAAAQ==")
            if (err.number <> 0) then
                if (err.number = LICENSE_EXPIRED) then
                    StoreLicenseResult.innerHTML = "You just received a
                        license that is already expired. Maybe your
                        clock is set wrong. Check and try again."
                elseif (err.number = LICENSE INCONSISTENT) then
                    StoreLicenseResult.innerHTML = "You just received a
                        corrupt license. Check with the license
                        server."
                else
                    StoreLicenseResult.innerHTML = "Error:" &
                        CStr(hex(err.number)) & ":" & err.Description
                end if
            end if
        End Sub
    </Script>
</HEAD>
<BODY>
    <p align="center"><B>V1 License</B></p>
    <OBJECT classid=CLSID:760C4B83-E211-11D2-BF3E-00805FBE84A6 id=DrmStore>
        <EMBED MAYSCRIPT TYPE="application/x-drm" HIDDEN="true"
            LICENSE="AAEAAHUAAAB3IFadTI8UJy3PzB9yilDoxgf5DR
            jqL4NXqFkns7*!Z4jFwCPX!oCDS1pPTHhMcmhaVStId0dMS
            lY4V3RhUT09AEeTvPQpGNt!AJ5BE6tB4ZJ5tDQJo*bnTOnA
            xatFIYch72C8A04kdFz8ZK*!UTj52e4dIRkQkMBHXXnma4x
            e9KFZB3QypiOMM6LQFyPs0ViJGwAAAAAAJYUBwMWAAEAAD
            gAAADRQt0mNlnxj7as*ys3NSMJaaWViZC1PpnlLxYqUdqCM
            m2iPIlzxu4zm5xxu39qj47qy33j5mXGbpviYTFldxMwNRRS
            ckf6kyEdHDya3LyAc2NjDB8AAAAAAAAAAAAAAAAAAQAAOAA
            AAGN!793njE8kEVW*BhFk*W5xfYgP*ymWlfUQely7kQCMci
            !Q6wPkIhG9LfC2Z85Uf01UPGTZ7pNCns0OdMfy85CZ5ceKk
            C0KYaQK*OrdqAQNY2MMHwAAAGMAAAABAAAAAQ==">
    </OBJECT>
    <span id="StoreLicenseResult">You have received a v1 license.</span>
```

```
</BODY>
</HTML>
```

4.2 DRM Version 7 License Request Example

The following example shows a sample Digital Rights Management (DRM) License Acquisition Protocol version 7 license request that contains all of the required elements. The fields might not contain complete data. For a complete description of an element, see the respective element topic.

```
<LICENSEREQUEST version="2.0.0.0">
  <V1CHALLENGE>

  <!--  DRMv1 challenge with empty KEYID string -->

  </V1CHALLENGE>
  <ACTIONLIST>

  <!--  Application is requesting the right to play the content. -->
  <!--  More than one right can be requested. -->

    <ACTION>Play</ACTION>
  </ACTIONLIST>

  <!-- Information about the client that is
       requesting the license. -->
  <CLIENTINFO>
    <CLIENTID>
    <!-- Client ID structure -->
    </CLIENTID>

    <!-- Version of the DRMv7 client -->
    <CLIENTVERSION>9.00.00.2778</CLIENTVERSION>

    <!-- Application security of client application -->
    <APPSECURITY>2000</APPSECURITY>

    <!-- Certificate subject ID of the component
         that is talking to DRM -->
    <SUBJECTID1>212</SUBJECTID1>

    <!-- Certificate subject ID of the component
         that is talking to the component-->
    <!-- talking to DRM. -->
    <SUBJECTID2>1107</SUBJECTID2>

    <!-- Version of Kernel Mode DRM on the client computer -->
    <DRMKVERSION>2.2.0.0</DRMKVERSION>
  </CLIENTINFO>


  <!-- WMRMHEADER section, verbatim from the header of the content -->

  <WMRMHEADER version="2.0.0.0">
    <DATA>
```

```
      </DATA>
      <SIGNATURE>

      </SIGNATURE>
    </WMRMHEADER>

</LICENSEREQUEST>
```

4.3 DRM Version 7 Nonsilent License Response Example

The following example shows a response Web page containing an encoded license.

```
<HTML>
<HEAD>
<TITLE></TITLE>
  <Script Language="JavaScript">

  function StoreV2License(hr)
  {
    netobj.StoreLicense("AAEAAHUAAADnIFW4Ec2j0JXEId5cfdhQoXCZJSPIjaKE
                         5L!FlSp0YM!7pSCayfFHVDBlTnRRS2tqa09GZENpcW54
                         akhnZz09AMnjjoZs5X9ZjuZCvFGDfSymhnp29w!0v0u9
                         t!NLeS5mw0I!iDNHqX0T5pZ0ie8HxJqQ23WRU1zOp*p8
                         OreBn3L1NzR2qaqJwSIP97XtS04mEwAAAAAAAJYUBQYQ
                         AAEAADgAAAAB2ZQI!btK6AOOJI68EEuHnnfVDsPjufRe
                         9FseC8IsW14EnD1HgjkJQ3*VKD9zKJB3oJQ9ZnbtJ10u
                         kgWxZtc5NwkIMU85AR8Aj6y0IcZpIxQFCBQAAG!JAAAA
                         AgAAqLkAAQAAOAAAADmkObY2USONnrxHr140bmTk!T9n
                         o7hB7EibVZl463LmVqQkywubKQ4l8RM!RwonN23ygvlh
                         efHw0BG2IAyFJ0GFxaThS1yagLYrZnxWSkc6FAYDAQAA
                         AAoAAAABAAAAAQ==");
  }

  </Script>
  </HEAD>
  <BODY onLoad="StoreV2License()">
  <OBJECT classid=clsid:A9FC132B-096D-460B-B7D5-1DB0FAE0C062
          height=0 id=netobj width=0 VIEWASTEXT>
  <EMBED MAYSCRIPT TYPE="application/x-drm-v2" HIDDEN="true">
  </OBJECT>

  </BODY>

</HTML>
```

4.4 DRM Version 7 License Example

The following code is a sample Digital Rights Management (DRM) License Acquisition Protocol version 7 license that contains all of the required elements. The values in each element might not be valid. For a complete description of an element, see the respective element topic.

```
<?xml version="1.0"?>
<LICENSE version="2.0.0.0">
   <LICENSORINFO>
      <DATA>
         <LID>{00000000-0000-0010-8000-00AA006D6D6D}</LID>
         <KEYID>asdfasdfas==</KEYID>
         <ISSUEDATE>20000102 23:20:14Z</ISSUEDATE>
         <CONTENTPUBKEY>jkjkjkjkjkjkjkjk==</CONTENTPUBKEY>
         <PRIORITY>15</PRIORITY>
         <META>
            <NAME>License for Contoso</NAME>
            <DESCRIPTION>License to play on pc and burn CD
            </DESCRIPTION>
            <TERMS>This license non-transferable</TERMS>
            <TRANSACTIONID>12344</TRANSACTIONID>
            <LICENSORNAME>Contoso</LICENSORNAME>
            <LICENSORSITE>www.Contoso.com</LICENSORSITE>
         </META>
         <ONSTORE>
           <CONDITION>
              <![CDATA[
                versioncompare(drm.version, "2.0.0.0") >= 0
              ]]>
           </CONDITION>
           <ACTION>
              <![CDATA[
                !exists(secstate.playcount)?(secstate.playcount=5;
                secstate.cdcachecount=1;1):1
              ]]>
           </ACTION>
         </ONSTORE>
         <ONSELECT>
           <CONDITION>
              <![CDATA[
               machine.datetime <= #19991231 09:00:00Z# &&
                content.CID==
                        "{00000000-0000-0010-8000-00AA00AAAEAE}"
                && app.minseclevel >= 500
              ]]>
           </CONDITION>
         </ONSELECT>
         <ONCLOCKROLLBACK>
           <ACTION>
              <![CDATA[
                deletelicense()
              ]]>
           </ACTION>
         </ONCLOCKROLLBACK>
         <ONACTION type="Play">
           <CONDITION>
              <![CDATA[
                secstate.playcount > 0
              ]]>
           </CONDITION>
           <ACTION>
              <![CDATA[
                secstate.playcount--;
              ]]>
```

```
      </ACTION>
    </ONACTION>
    <ONACTION type="Print.Redbook">
       <CONDITION>
          <![CDATA[
            app.seclevel >= 1000 &&
                secstate.cdcachecount > 0
          ]]>
       </CONDITION>
       <ACTION>
          <![CDATA[
            secstate.cdcachecount--;
          ]]>
       </ACTION>
    </ONACTION>
    <ONACTION type="Backup">
       <CONDITION>
          <![CDATA[
            1
          ]]>
       </CONDITION>
    </ONACTION>
    <ONACTION type="Restore">
       <CONDITION>
          <![CDATA[
            1
          ]]>
       </CONDITION>
    </ONACTION>

    <ENABLINGBITS>
      <ALGORITHM type="MSDRM" />
      <PUBKEY type="machine">JKJKJKJK==</PUBKEY>
      <VALUE>AAAABBBBAAAADDDD</VALUE>
      <SIGNATURE>asdfasdfd==</SIGNATURE>
    </ENABLINGBITS>
    <CONTENTREVOCATION>
       <DATA>
          <SEQUENCENUMBER>1</SEQUENCENUMBER>
          <CONTENTPUBKEY>pokpokpokkk</CONTENTPUBKEY>
          <LICENSESERVERPUBKEY>assdsdfdfdfdf==
          </LICENSESERVERPUBKEY>
          <CONDITION>
             <![CDATA[
               deletelicense();0
             ]]>
          </CONDITION>
      </DATA>
      <SIGNATURE>
          <HASHALGORITHM type="SHA"/>
          <SIGNALGORITHM type="MSDRM"/>
          <VALUE>SDSDSDSDSDSDSD</VALUE>
      </SIGNATURE>
    </CONTENTREVOCATION>
  </DATA>
  <SIGNATURE>
    <HASHALGORITHM type="SHA">
    <SIGNALGORITHM type="MSDRM">
```

```
            <VALUE>AAAAAAAAAAAAAA</VALUE>
         </SIGNATURE>
         <CERTIFICATECHAIN>
            <!- The first one is the cert issued
                by the root authority -->
            <!- The last one is the one issued
                to the license server -->
            <CERTIFICATE>JNKNKNKN</CERTIFICATE>
            <CERTIFICATE>HBHBHBHB</CERTIFICATE>
         </CERTIFICATECHAIN>
      </LICENSORINFO>
   </LICENSE>
```

4.5 DRM Version 11 License Example

The Digital Rights Management (DRM) License Acquisition Protocol version 11 license is identical to the Digital Rights Management License Acquisition Protocol version 7 license.

5 Security

The following sections specify security considerations for implementers of the Digital Rights Management License Acquisition Protocol.

5.1 Security Considerations for Implementers

None.

6 Appendix A: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows Vista - Applies to versions 1, 7, and 11.
- Windows Server 2003 - Applies to versions 1 and 7.
- Windows XP - Applies to versions 1, 7, and 11.
- Windows 2000 - Applies to versions 1 and 7.
- Windows NT - Applies to version 1.

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 2.1: Transport Mechanism. Implemented in Windows XP SP2, Windows Vista, Windows 2000, Windows Server 2003 and Windows Server 2008. The Digital Rights Management License Acquisition Protocol uses HTTP and HTTPS for data transfer. The TCP ports are configurable by the implementer.

7 Index

W